Real-Time Hazard Symbol Detection and Localization Using UAV Imagery

Conference Paper · September 2017

CITATIONS
0

READS
244

3 authors, including:

Nils Tijtgat
Ghent University
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Bruno Volckaert
Ghent University
95 PUBLICATIONS **469** CITATIONS
SEE PROFILE

Real-Time Hazard Symbol Detection and Localization Using UAV Imagery

Nils Tijtgat*, Bruno Volckaert†, Filip De Turck‡
Authors are with Ghent University - imec, IDLab, Department of Information Technology
Zwijnaarde-Technologiepark 15 B-9052 Zwijnaarde, Belgium
Email: {nils.tijtgat, bruno.volckaert, filip.deturck}@ugent.be

***Abstract*—Unmanned Aerial Vehicle (UAV) technology is advancing at a fast pace following the strong rise in interest in its applications for a wide variety of scenarios. One of the promising use cases for UAVs is their deployment during emergency and rescue operations. Their high mobility, aerial viewpoint and flexibility to be operated autonomously are huge assets during crises. UAVs exist in a wide range when it comes to cost, and so do the sensors or accessories they can carry along as payload. During emergencies it may be beneficial to have several low-cost UAVs on site, as opposed to one highly sophisticated, fully-equipped aerial platform. This way more ground can be covered in a shorter span of time during search-and-rescue operations. The situation where a single UAV is on the ground to recharge its battery can also be avoided. In this paper, we present an architecture that identifies and locates objects of interest in real-time using low-cost hardware and a state of the art object detection algorithm. We avoid the use of expensive LIDAR sensors and UAVs, but still manage to determine the position of specific predefined objects in the field with high precision. Without augmenting our detection setup by using intelligent adaptive flight patterns, we can locate an object within 1.5m of its actual location at very low latency. Our detection chain delivers the result and its location in well under one second.**

I. Introduction

Unmanned Aerial Vehicles (UAVs), also known as Remotely Piloted Aircraft Systems (RPAS) or referred to by the general public quite simply as 'drones', are rapidly gaining interest on a global scale. Though still in its early phase, the rapid development of technology, miniaturization of sensors and computational platforms and accompanying drop in cost are leading to an accelerated adoption of this new type of technology. UAVs can be divided into two categories: the fixed wing and multi-rotor designs. Both types have inherent advantages and limitations. Capable of longer flight times (many devices get up to 45 minutes), wing-type UAVs excel at aerial surveys over large areas. Their multi-rotor variants are capable of VTOL (Vertical Take-Off and Landing) and boast the great feature of being able to hover in place at a certain location. Their flight time (in general around 25 minutes for small UAVs) is reduced compared to fixed wing designs. No matter the type of UAV, their operational time is limited when it comes to being viable solutions for certain applications. Having an 'eye in the sky' for only 20 minutes before coming in for a fresh charge is not ideal in calamity situations, or during remote inspections. It also rarely occurs at emergency scenes that only one event is taking place that requires monitoring. Multiple UAVs on scene could prove a sensible solution, at a linear increasing cost in function of the number of UAVs. Even though many complex aerial platforms are becoming increasingly expensive, interesting events are taking place at the low-end of the market. As costs have gone down recently, affordable consumer drones feature full HD cameras, flight times comparable to their more expensive variants and an ever more impressive feature set. It's interesting to research if these devices could cooperate at solving certain tasks faster than one expensive UAV is capable of. Prime example of this is the detection of hazard icons at an emergency scene.

To that end, this paper introduces a first proof of concept implementation of one of the modules of the 3DSafeGuard-VL project, that aims at supporting decision makers during emergencies by deploying UAVs at disaster sites. We will demonstrate the architecture and performance of this first module, namely the algorithms that detect hazard symbols from a drone video feed, and determine where the detected object is currently located. This position can then be used on a real-time crisis map, and serve as input to the decision support system that 3DSafeGuard-VL will embody.

II. Related Work

Multiple publications focus on detecting objects or persons using UAVs, often making use of thermal imagery. Gszczak et al. [1] demonstrate a setup that detects vehicles and persons. They combine trained cascaded Haar classifiers with secondary confirmation using a thermal camera. Overall episodic object detection rate is stated to exceed 90%. Images are processed at 1Hz, severely limiting the maximal speed a UAV can achieve while scanning an area in order not to miss anything. Another limiting factor is the requirement of having an expensive thermal camera on board. Rudol et al. [2] have the same goal, a comparable setup (two onboard cameras and Haar classifier based detection) but can detect humans at a rate of up to 25fps and run the image processing on the UAV in-flight.

Other approaches focus on detecting and tracking moving objects in UAV imagery [3][4]. These methods detect and track objects from an aerial platform using the onboard camera. They build on existing motion tracking solutions, and make them more robust for use with UAV video. Both solutions are not very suitable for our application, as they are limited to detecting moving objects only.

Sokalski et al. [5] provide an interesting solution for search-and-rescue operations by performing salient detection. The approach is targeted at finding objects that differ from their environment in rural, uncluttered and relatively uniform areas however, making it unsuited to apply in urban environments. Detection of specific predefined objects is not a possibility: every object that differs from its environment is detected since its detection method is based on a saliency map generated by UAV imagery of the area. We want to detect only what we instruct it to, and know exactly what object was detected.

As completing reference: [6] demonstrates how a fully autonomous drone, targeted at indoor and outdoor urban search and rescue operations could look like. It features a laser sensor and stereo camera among its sensor package and is completed with onboard processing. Unfortunately, its computer vision performance is not explained in detail. The 1.6GHz Intel Atom hardware seems underpowered to perform any serious object detection. Hulens et al. present

an interesting reference on how to choose the ideal onboard processing unit for UAV applications in this regard [7]. For deep learning based computer vision requiring GPU power, the solutions presented there may still lack in performance. The recently released NVIDIA Jetson TX2[1] promises to bring a whole new level of performance to embedded devices and UAVs. Vega et al. [8] show that its predecessor (Jetson TK1) already poses a step up from standard embedded hardware.

In conclusion: to our knowledge, relevant research combining novel deep learning computer vision approaches with low cost UAV devices to perform real-time object detection is non-existent. Many relevant publications use expensive hardware or outdated computer vision algorithms, differing from our solution.

III. Methodology

The entire object detection and position determining setup consists of multiple different modules working together. Figure 1 illustrates the global architecture and the different components it is composed of. The following subsections will take a closer look at every module.

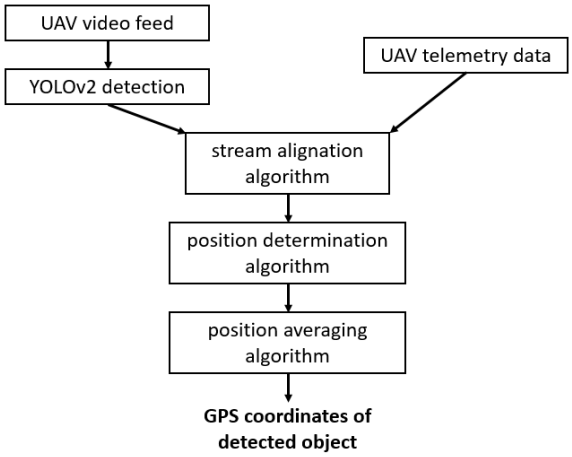

Fig. 1. System architecture

A. YOLOv2 Real-Time Object Detection

Ever since the major breakthrough of deep neural networks, general computer vision object detection can attain a very high detection rate [9]. A remaining, non-trivial task is the real-time execution of this detection, using a live video feed for instance. At the moment, there are only a few algorithms capable of this feat (detection faster than 24fps). The first real-time object detection algorithm is DPM V5 [10], that runs at either 30fps or 100fps (with strongly declining detection rates at higher frame rates). A second one is YOLOv2 [11], proposed by Redmon et al, capable of an impressive framerate of 40fps (varying with available hardware) while still maintaining a high detection rate of 78.6 mean average precision (mAP) on the VOC 2007 dataset. Its direct competitor Faster R-CNN [12], often cited as state-of-the-art object detection algorithm has a lower score on the same VOC 2007 dataset and is slower (lower framerate and higher latency). Comparing these different algorithms clearly indicates YOLOv2 as the winning algorithm for our application. The much higher frame rate and better detection rate are compelling arguments to select this algorithm for our real-time object detection.

[1]https://developer.nvidia.com/embedded/buy/jetson-tx2

In the final implementation, a wide variety of hazard pictograms should be detected by our algorithm. In this paper, we present a proof of concept of the entire processing chain with detection limited to one pictogram. We have selected the NFPA 704 'fire diamond' pictogram[2] (see Figure 2) as the target of our object detection algorithm. We let YOLOv2 train on a diverse dataset composed of 282 training images that include the NFPA pictogram. The exact method we used to train YOLOv2 falls out of the scope of this paper, we do however provide an online article explaining the steps in detail[3]. Our training set contains a combination of real world photos (sourced using both online image search and our own drone flights) and images generated using the Unreal Editor, a software suite targeted at the development of games, simulations and visualizations. The last source of training images deserves extra attention. We use photorealistic Unreal 3D environments extensively to create images that are non-trivial to collect. An example is the NFPA sign in different extreme weather situations: rain or strong sunlight changes the appearance of the pictogram, but our algorithm should still detect it. Adding these valuable images to our training set could be difficult if it doesn't rain often, or if our development drone is not water resistant. Unreal can render perfectly valid training images, saving time and effort when composing the training set. A realistic 3D model of our object is a requirement of course. The concept of using photorealistic 3D environments for deep learning is not limited to our use case: the Unreal engine has been used in different deep learning applications before [13][14], and as environments and objects become more and more realistic, will probably be used even more in the future.

One of the shortcomings of YOLOv2, is that it struggles to find small images, an inherent consequence of the 'only look once' approach of the algorithm. Both Faster R-CNN and SSD pass over an image multiple times with different detection sizes and are hence more robust in this regard. To counter this YOLOv2 behaviour, we have added many test flight images where the fire diamond is very small or far away to our training set. A further suggestion to improve small object detection using YOLOv2, is to increase the the height and width of the detection screen (input layer of the neural network) from 416x416 (size used when training) to higher multiple of 16 (608x608 or 832x832 for instance) during detection [11]. In our case however, this negatively impacted our false positive detection by a big amount, so we chose not to apply this technique.

Fig. 2. NFPA 704 'fire diamond'

B. Distance Estimation

Distance estimation is an important part of our algorithm, as its output is required to correctly determine the GPS coordinates of the detected object. Extracting a 3D depth model from a single image without the aid of stereoscopic cameras is an interesting area, one where recently a lot

[2]http://www.nfpa.org

[3]https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/

of novel approaches to the subject have been published. A first approach involves applying supervised learning to a depth-image dataset, as presented by Saxena et al. [15]. More recently, Chahal, Pippal et al. [16] propose the use of a combination of machine learning techniques to generate a depth image from a single image. Albeit the latter displaying promising results, no quantification is made as to how long it takes to process one image. Judging from the architecture, odds are it will not run in real-time on a video feed without employing a complicated distributed setup. Possibly even worse: it may need training for specific backgrounds or environments meaning it is not universally applicable without prior configuration/training. As a last complication: in a depth image calculated this way, there is no reference as to how far away an object is from the camera in an absolute way. One can only conclude that object A is closer to, or further away from the camera than object B using this technique. Determining what the distance from A to camera is, cannot be done decisively.

To overcome these complications, we revert to a simplified approach for this proof of concept implementation. This choice can further be motivated in our case by the fact that a highly accurate distance result is not of vital importance, but detection time latency is. The location accuracy can easily be increased by letting the drone navigate closer to where a detection was made: as the drone gets closer, the error introduced by measuring from a distance becomes smaller. During emergencies it's more important to immediately detect a dangerous goods container on the premises with fairly accurate positioning, as opposed to detecting it too late and endangering ground staff.

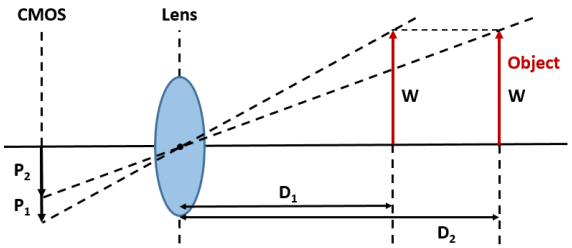

Fig. 3. Lens system

Our simplified approach is based on the triangle similarity (see Figure 3): if we calibrate our optical sensor with an object of known size and known distance to the camera as a reference it allows us to easily find the depth of an object in an image. Following equation describes the situation at hand:

$$F = \frac{P \cdot D}{W}$$

where F is the focal length of the camera, P is the perceived dimension in pixels of the object, D the distance to - and W the actual width of our object.

Calibration is as simple as processing the image of a reference object (having a width of 1m for instance) at an accurately measured distance to the camera. Solving the equation yields the focal length of our system, which we can then use to calculate the distance to a detected object. To calibrate our system, we use a basic specific pattern that our software can recognize as the reference calibration object. This fairly uncomplicated approach yields surprisingly good results, and what it may lack for in absolute positional accuracy, it makes up for in speed and reduced architectural complexity.

C. UAV Telemetry

A vital component of our system is the reliable retrieval of UAV flight data (position, heading, airspeed). This must be perfectly in sync with the video detection feed to assure accurate positioning results. A time delta between drone position and video detection of only 1 second could already have detrimental effects on our results: UAVs are capable of moving at multiple meters per second and can change their heading very fast. It is thus vital to keep these two tightly synchronized. To implement the proof of concept solution presented in this paper, we have opted not to consider a real-time streaming solution to eliminate any timing problems that may arise in this case: proving that the setup can work reliably is our primary goal. The tests we did for our research were hence 'offline': both video feed and telemetry data were collected after the flight, and replayed to simulate an 'online' environment.

D. AirSim

Earlier on, we already discussed the benefits of using a photo-realistic 3D environment as a tool to help train and implement our solution. Not only visual data can be rendered using the Unreal engine, recently Microsoft has published their open-source AirSim Unreal simulator [17]. This state of the art UAV simulator provides a very realistic (both physical and visual) environment to test UAV software applications in real-time. It boasts both a hardware-in-the-loop (HIL) as a software-in-the-loop (SITL) configuration. Implementations tested in AirSim can thus be seamlessly integrated on an actual Pixhawk PX4 based UAV. Being a work in progress, it is targeted at supporting machine learning applications and already supports many different features. We have used AirSim to test our YOLOv2 object detection algorithm in many different situations, helping us uncover issues and fine-tune our detection model before taking to the skies outdoors. AirSim offers an API to expose most of the relevant UAV data and an accurate model for the physical environment. We are currently implementing our entire configuration (Figure 1) in AirSim for offline simulation and expansion of our total number of testing environments.

E. Stream Joining

As mentioned before: attaining a high quality synchronization between UAV telemetry and video detection feed is vital for the performance of our algorithm. An important factor in this situation is the fluctuating frame rate YOLOv2 attains. As an example: on one development system running a somewhat outdated NVIDIA GTX670 supplied with a constant 15fps MP4 video we measured frame rates between 13,9fps and 17,5fps (as reported by YOLOv2 itself). To combat this offset, and be more robust against whatever may temporarily influence the frame rate even more, we report the frame number at detection time. In our offline case this works very well, since UAV telemetry data and video feed start at the exact same time. As soon as we advance to an online streaming solution, we will have to rethink and improve this approach, especially because the frame rate of the source video feed will no longer be stable. Multiplexing data frames containing timing information may prove the preferred solution. In any case: even our offline situation suffers from data imperfections. Our stream joining algorithm matches detections to their respective telemetry data points and filters out invalid matches. We filter out UAV data containing imprecise GPS data, incorrect heading information or data points having a

time difference greater than 150ms from detection time. Detections with a probability lower than 40% are also ignored. Matches that pass these tests are forwarded to the position determination algorithm.

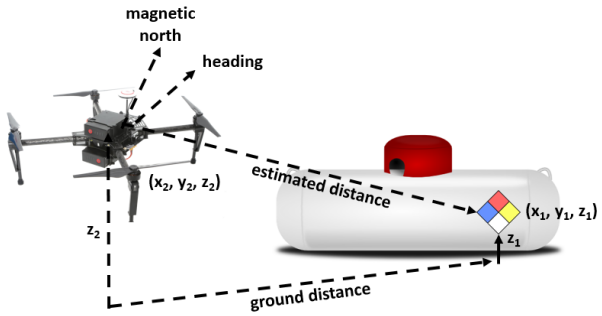

Fig. 4. Determining the position of a detected object

F. Position Determination

The matched tuples (*detection data*, *UAV data*) this algorithm receives are used to determine the actual position of our detected object. We need the exact position the observation was made (UAV GPS position), the distance to the object (as estimated using the detection bounding box dimensions) and the heading to the object. Figure 4 illustrates this. Following formulas are used to calculate the resulting position given previously described variables:

$$\phi_2 = \arcsin(\sin\phi_1 \cdot \cos\delta + \cos\phi_1 \cdot \sin\delta \cdot \cos\theta)$$
$$\lambda_2 = \lambda_1 + \operatorname{atan2}(\sin\theta \cdot \sin\delta \cdot \cos\phi_1, \cos\delta - \sin\phi_1 \cdot \sin\phi_2)$$

where ϕ is latitude, λ is longitude, θ is the bearing (clockwise from north), δ is the angular distance d/R, d the distance, R the earths radius (6371km)

Determining the bearing is the result of combining the UAV heading and the position at which the object is detected in the image. Figure 5 visualizes this thought. The horizontal field depends on the physical construction of the camera, the algorithm uses a value specific for the camera used. A first approach is to use the detected x-coordinate and take the relative part of the horizontal field of view (as indicated in the image). This is of course not accurate enough as the angle α greatly varies with object distance to camera. Our current implementation therefore also takes into account the y-position (objects that are higher up in the image tend to be further away, with a resulting smaller angle α) and the size of the bounding box (smaller bounding box, smaller angle α). The algorithm we currently use is not a definitive robust solution, but results in a general approximation of the bearing, enabling us to test the entire detection setup. Further research is required to improve the determination of α since not all the assumptions we made are valid under every circumstance.

Our algorithm delivers both an instantaneous detection position, and an averaged result over time. Since the objects we currently aim to detect rarely move around dynamically, we average over the entire flight time. If displaceable objects (such as barrels with hazard symbols on them) need to be considered as well, a windowed approach using a shorter averaging time can compensate for this.

IV. Results

We have implemented the architecture as described, and validated its performance using a Parrot Bebop 2 UAV. Although being a low cost device, it's perfectly up to the task and supports our use case of having multiple low cost UAVs on site to scan an area rather than having one expensive aerial platform in the air. The FlightData Manager software tool[4] allows us to easily collect the UAV telemetry data and convert it to geolocation formats (*.kml) or simple *.csv files. The Bebop OS can be reconfigured to log data at a much higher rate (200Hz)[5] if required. To create a fixed validation GPS location, we position ourselves right next to the NFPA pictogram to be detected while flying the UAV (see Figure 6). The Bebop 2 controller logs its GPS location when connected to the Android/iOS app, allowing us to accurately validate the position determining algorithm. The test setup consists of a NFPA 704 pictogram (20cm edge) mounted on a plate right below the UAV controller. This way, we know the exact GPS position of our detection target.

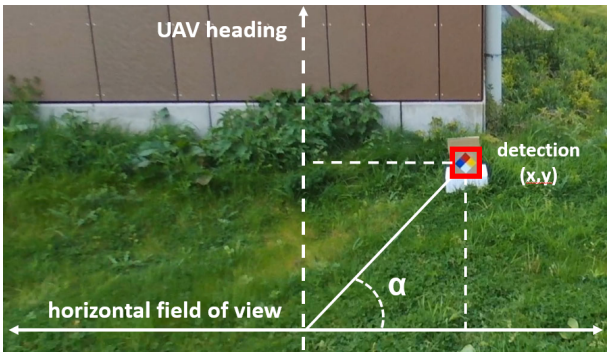

Fig. 5. Determining the bearing using heading and detection position

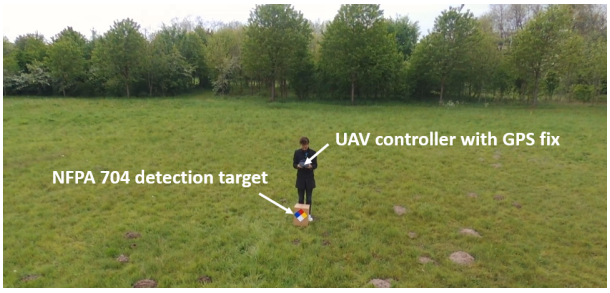

Fig. 6. Detection validation setup

We fly the UAV around in a semi random pattern to scan the area and let our software determine the pictographs location. The distance between averaged detection and actual object position is then evaluated. To monitor and inspect the detection, we wrote a small graphical interface to visualize the different actors and detections as the algorithm runs. Figure 7 illustrates this tool.

A demonstration video of the entire test case is available on our GitHub page[6]. In the demonstration video, our algorithm determines the position of the object 1.2091m away from the actual pictogram after flight completion. Given that GPS positioning in general has an accuracy between 3 to 5 meters this is a very acceptable result. Most of the time the objects with a hazard symbol on them tend to be fairly big (gas tank, container), making the result even more satisfactory.

We tested our implementation on a machine equipped with

[4]https://sites.google.com/site/pud2gpxkmlcsv/

[5]https://github.com/nicknack70/bebop/raw/master/UBHG/UBHG1_6_1.pdf

[6]https://timebutt.github.io/Hazard-Symbol-Detection/

Fig. 7. drone C controller D detection A average detection

Fig. 8. Bounding boxes cutting off edges of NFPA pictogram

an Intel i7-6700K CPU and NVIDIA GTX 1080Ti GPU (11.3TFLOPS). A frame rate of 40fps was achieved when processing a single video stream, GPU load was around 42% during this run. We conducted a second test, running two parallel video feeds into our algorithm and found both streams to run at just over 35fps, GPU load close to 65%. We had to double the YOLOv2 subdivisions parameter to keep the memory CUDA requires below the available 11Gb VRAM. Regarding the performance of our algorithm: the indicated frame rates are attained without visualization of the processed video feed. We can use OpenCV to display the resulting video, but this drops the frame rate to just under 22fps (in the demonstration video the frame rate is closer to 19fps, this discrepancy can be attributed to the screen capture software running). When visualizing two parallel streams, the frame rate fluctuates slightly around 20fps.

It is non-trivial to quantify the detection performance of our trained model, as there is no standardized data set available for the NFPA 704 pictogram. Showcasing the performance by means of the demonstration video is currently the only validation we can provide. We do note, that the algorithm sometimes struggles to fit the bounding box correctly (see Figure 8): its dimensions fluctuate around the actual pictogram as the bounding box tends to cut off the triangle edges, especially when the pictogram is viewed under an angle. As YOLOv2 seems to consistently underestimate the pictograph's dimensions, we compensate for this in the distance estimation algorithm. A proposed measure to correct this erroneous behavior, is to further train the model with an increased amount of actual UAV images in different situations and from multiple angles. Aerial imagery currently represents about 20% of our total training set.

V. Future work

A great way of increasing the position determining accuracy is to augment the detection setup with an intelligent autonomous flight control algorithm that navigates the UAV closer to detected objects. Flying nearer to the object will decrease errors introduced by the limited resolution and angle estimation, and present more data to the averaging algorithm to smooth out earlier and more incorrect detections. The implementation of a custom UAV telemetry retrieval application will allow us to test the setup in-flight.

VI. Conclusion

This paper reports on findings of the first module of the 3DSafeGuard decision support engine, capable of detecting pretrained objects in a live UAV video feed and determining the location of this detected object in real-time. Our setup currently supports one UAV, during large emergencies this may no longer suffice as more UAVs could be required on-scene. We have shown that our algorithm can process two parallel video feeds at 35fps using adequate hardware, as a first step in this direction.

May 08, 2017

Acknowledgment

The authors would like to thank the other partners in the 3DSafeGuard-VL project and the Agency of for Innovation by Science and Technology (Vlaio) for funding and support. In loving memory of Ann Vroman.

References

[1] A. Gaszczak, T. P. Breckon, and J. Han, "Real-time people and vehicle detection from uav imagery," in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2011, pp. 78 780B–78 780B.

[2] P. Rudol and P. Doherty, "Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery," *2008 IEEE Aerospace Conference*, 2008.

[3] G. R. Rodríguez-Canosa, S. Thomas, J. del Cerro, A. Barrientos, and B. MacDonald, "A real-time method to detect and track moving objects (datmo) from unmanned aerial vehicles (uavs) using a single camera," *Remote Sensing*, vol. 4, no. 4, pp. 1090–1111, 2012.

[4] M. Teutsch and W. Krüger, "Detection, segmentation, and tracking of moving objects in uav videos," in *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*. IEEE, 2012, pp. 313–318.

[5] J. Sokalski, T. P. Breckon, and I. Cowling, "Automatic salient object detection in uav imagery," in *Proc. 25th International Conference on Unmanned Air Vehicle Systems*. Citeseer, 2010, pp. 11–1.

[6] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[7] D. Hulens, J. Verbeke, and T. Goedemé, "Choosing the best embedded processing platform for on-board uav image processing," in *International Joint Conference on Computer Vision, Imaging and Computer Graphics*. Springer, 2015, pp. 455–472.

[8] A. Vega, C.-C. Lin, K. Swaminathan, A. Buyuktosunoglu, S. Pankanti, and P. Bose, "Resilient, uav-embedded real-time computing," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 736–739.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] M. A. Sadeghi and D. Forsyth, "30hz object detection with dpm v5," in *European Conference on Computer Vision*. Springer, 2014, pp. 65–79.

[11] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[13] W. Qiu and A. Yuille, "Unrealcv: Connecting computer vision to unreal engine," in *Computer Vision–ECCV 2016 Workshops*. Springer, 2016, pp. 909–916.

[14] A. Lerer, S. Gross, and R. Fergus, "Learning physical intuition of block towers by example," *arXiv preprint arXiv:1603.01312*, 2016.

[15] A. Saxena, S. H. Chung, and A. Y. Ng, "3-d depth reconstruction from a single still image," *International journal of computer vision*, vol. 76, no. 1, pp. 53–69, 2008.

[16] N. Chahal, M. Pippal, and S. Chaudhury, "Depth estimation from single image using machine learning techniques," in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*. ACM, 2016, p. 19.

[17] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," 2017.